## horizontal line



DeepDSL Implementation

IUST Compiler Design - 4031

─

Group 10

400522292, 400522175, 400481117, 400522112

# Overview

هدف این پروژه، پیاده سازی DeepDSL با استفاده از ANTLR برای تعریف گرامر و تولید کد به زبان پایتون است. این پروژه ابزار قدرتمندی برای تولید خودکار کدهای مرتبط با یادگیری عمیق فراهم خواهد کرد .قواعد گرامری این پروژه با استفاده از ابزار ANTLR پیاده شده است. کدهای مربوطه و در نهایت کد تولید نیز به زبان پایتون خواهد بود.

# Goals

به صورت کلی در این پروژه، هدف طراحی و پیاده‌سازی یک شبکه عصبی با استفاده از دستورالعمل‌های زبان گرامری تعریف‌شده در deepDSL می‌باشد. این گرامر برای توصیف ساختار شبکه عصبی، آموزش، ارزیابی، و پیش‌پردازش داده‌ها طراحی شده است. در ادامه، ساختار کلی پروژه و توضیحاتی در خصوص هر بخش آمده است.

# Specifications

## Phase 1: Grammar:

(network):

پروژه با معرفی یک شبکه عصبی شروع می‌شود که شامل مشخصات مختلف از جمله لایه‌ها، تنظیمات آموزش، داده‌ها، و فرآیندهای ارزیابی و تجسم است. این بخش با استفاده از شناسه شبکه و براساس اطلاعات تعریف‌شده، شامل لایه‌ها و تنظیمات مختلف می‌باشد

(layer):

هر لایه در شبکه عصبی شامل ویژگی‌های خاصی از جمله نوع لایه (Dense یا Flatten)، تعداد واحدها، تابع فعال‌سازی، و شکل ورودی است. برای هر لایه، نوع آن به همراه تعداد واحدهای موجود در آن تعریف می‌شود. همچنین در صورت نیاز، تابع فعال‌سازی و ابعاد ورودی نیز مشخص می‌شوند.

(training):

در این بخش، تنظیمات مرتبط با فرآیند آموزش شبکه عصبی شامل انتخاب بهینه‌ساز (مانند Adam، SGD یا RMSProp)، تابع خطا (مانند Sparse Categorical Cross entropy یا Mean Squared Error)، معیارهای ارزیابی (مانند دقت یا از دست دادن)، تعداد دوره‌های آموزشی (epochs)، اندازه دسته (batch size)، و نسبت تقسیم داده‌ها برای اعتبارسنجی (validation split) مشخص می‌شود.

(dataset):

مشخصات مجموعه داده برای آموزش شبکه عصبی نیز در این بخش تعریف می‌شود. شامل منبع داده و روش‌های پیش‌پردازش مورد استفاده بر روی داده‌ها است. پیش‌پردازش معمولاً شامل عملیات نرمال‌سازی داده‌ها است.

تجسم (visualize):

در این بخش، تنظیمات مربوط به تجسم داده‌ها و نتایج شبکه عصبی شامل ساخت گرید برای تجزیه و تحلیل نتایج پیش‌بینی شبکه است.

ارزیابی (evaluate):

در این بخش، معیارهای ارزیابی شبکه برای تحلیل عملکرد آن در حین فرآیند ارزیابی مشخص می‌شود.

grammar deepDSL;

*// Parser Rules*

start: network EOF;

network: 'network' ID '{'

layer+

training

(dataset)?

(visualize)?

(evaluate)?

'}';

layer: 'layer' ID '{'

types

units

(activation)?

(input\_shape)?

'}';

training: 'training' '{'

optimizer

loss

metric\_choice

epochs

batch\_size

validation\_split

'}';

dataset: 'dataset' ID '{'

source

preprocessing

'}';

visualize: 'visualize' '{'

'grid' ':' '[' value ',' value ']' ';'

'}';

evaluate: 'evaluate' '{'

metric\_choice

'}';

optimizer: 'optimizer' ':' optimizer\_func ';';

optimizer\_func: ('adam' | 'sgd' | 'rmsprop');

loss: 'loss' ':' loss\_func ';';

loss\_func: ('SparseCategoricalCrossentropy' | 'MeanSquearedError');

metric\_choice: 'metric' ':' '[' metrics (',' metrics)\* ']' ';';

metrics: ('accuracy' | 'loss');

epochs: 'epochs' ':' value ';';

batch\_size: 'batch\_size' ':' value ';';

validation\_split: 'validation\_split' ':' value ';';

source: 'source' ':' path ';';

preprocessing: 'preprocessing' ':' preprocessing\_func ';';

preprocessing\_func: 'normalize' value;

types: 'type' ':' type ';';

units: 'units' ':' value ';';

activation: 'activation' ':' activation\_func ';';

activation\_func: ('Relu' | 'Sigmoid' | 'Softmax' | 'Tanh' | 'Linear');

input\_shape: 'input\_shape' ':' '[' value (',' value)\* ']' ';';

value: (INT | FLOAT);

path : STRING;

type: ('Dense' | 'Flatten');

*// Lexer Rules*

ID: [a-zA-Z\_][a-zA-Z\_0-9]\*;

INT: [0-9]+;

FLOAT: [0-9]+ '.' [0-9]+;

STRING: '"' .\*? '"';

WS: [ \t\r\n]+ -> skip;

COMMENT: '#' ~[\r\n]\* -> skip;

## Phase 2-1: Listener:

یک Listener سفارشی برای تجزیه‌گر گرامر deepDSL پیاده‌سازی شده است. هدف این Listener، پردازش درخت تجزیه (AST) با استفاده از قوانین گرامری موجود در deepDSL است. این Listener قادر است به طور خودکار برای هر قاعده موجود در گرامر، یک زیر درخت AST ایجاد کند و اطلاعات آن را ذخیره کند.

#### کلاس DeepDSLCustomListener

کلاس DeepDSLCustomListener از کلاس پایه deepDSLListener ارث‌بری می‌کند و چندین متد برای پیاده‌سازی منطق سفارشی خود دارد. این متدها با استفاده از make\_ast\_subtree، زیردرخت‌هایی را برای هر قاعده گرامری تولید می‌کنند. در اینجا شرح جزئیات پیاده‌سازی آمده است:

ساختار کلاس:

کلاس DeepDSLCustomListener شامل متغیرهایی همچون overridden\_rules است که لیستی از قواعد گرامری است که رفتار پیش‌فرض آنها تغییر کرده است.متغیر rule\_names برای نگهداری نام‌های قواعد گرامری استفاده می‌شود.ast شیئی از کلاس AST است که درخت تجزیه را مدیریت می‌کند.

متد exitEveryRule:

این متد برای پردازش هر قاعده که در درخت تجزیه یافت می‌شود، فراخوانی می‌شود.اگر قاعده در لیست overridden\_rules نباشد، زیر درخت مرتبط با آن قاعده ساخته می‌شود.

متدهای exit:

برای هر قاعده گرامری نظیر network, layer, activation, input\_shape, training, dataset, visualize, metric\_choice, evaluate و غیره، متد خاصی تعریف شده است.

این متدها با استفاده از make\_ast\_subtree زیردرخت‌هایی را برای هر قاعده ایجاد می‌کنند و به درخت تجزیه اضافه می‌کنند.در برخی موارد، گزینه keep\_node=True یا keep\_node=False تعیین می‌کند که آیا گره مربوط به آن قاعده در AST حفظ شود یا خیر.

from Repository.ast import AST

from Repository.make\_ast\_subtree import make\_ast\_subtree

from gen.deepDSLListener import deepDSLListener

from gen.deepDSLParser import deepDSLParser

class DeepDSLCustomListener(deepDSLListener):

def \_\_init\_\_(self, rule\_names):

self.overridden\_rules = [

'layer',

'activation',

'input\_shape',

'training',

'dataset',

'visualize',

'evaluate',

'metric\_choice',

]

self.rule\_names = rule\_names

self.ast = AST()

def exitEveryRule(self, ctx):

rule\_name = self.rule\_names[ctx.getRuleIndex()]

if rule\_name not in self.overridden\_rules:

make\_ast\_subtree(self.ast, ctx, rule\_name)

def exitNetwork(self, ctx):

make\_ast\_subtree(self.ast, ctx, ctx.getChild(1).getText())

def exitLayer(self, ctx):

make\_ast\_subtree(self.ast, ctx, "layer", keep\_node=True)

def exitTypes(self, ctx):

make\_ast\_subtree(self.ast, ctx, ctx.getChild(1).getText(), keep\_node=False)

def exitUnits(self, ctx):

make\_ast\_subtree(self.ast, ctx, ctx.getChild(0).getText(), keep\_node=False)

def exitActivation(self, ctx):

make\_ast\_subtree(self.ast, ctx, "activation", keep\_node=True)

def exitTraining(self, ctx):

make\_ast\_subtree(self.ast, ctx, "training", keep\_node=True)

def exitVisualize(self, ctx):

make\_ast\_subtree(self.ast, ctx, "visualize", keep\_node=True)

def exitInput\_shape(self, ctx:deepDSLParser.Input\_shapeContext):

make\_ast\_subtree(self.ast, ctx, "input\_shape", keep\_node=True)

def exitMetric\_choice(self, ctx:deepDSLParser.Input\_shapeContext):

ctx.compound = True

make\_ast\_subtree(self.ast, ctx, f"metric\_choice", keep\_node=True)

def exitDataset(self, ctx:deepDSLParser.Input\_shapeContext):

make\_ast\_subtree(self.ast, ctx, "dataset", keep\_node=True)

def exitSource(self, ctx:deepDSLParser.Input\_shapeContext):

make\_ast\_subtree(self.ast, ctx, ctx.getChild(0).getText(), keep\_node=False)

def exitPreprocessing\_func(self, ctx:deepDSLParser.Input\_shapeContext):

make\_ast\_subtree(self.ast, ctx, ctx.getChild(0).getText(), keep\_node=False)

def exitEpochs(self,ctx):

make\_ast\_subtree(self.ast, ctx, ctx.getChild(0).getText(), keep\_node=False)

def exitEvaluate(self, ctx:deepDSLParser.EvaluateContext):

make\_ast\_subtree(self.ast, ctx, "evaluate", keep\_node=True)

## Phase 2-2: Code Generation:

کد نهایی به صورت یک کد class-based است که ابتدا متد های مربوط به rule ها نوشته شده جنریت میشوند و در نهایت یک instance از كلاس ساخته شده و توابع مربوطه جنریت میشوند.

#### کلاس DeepDSLCodeGenerator

ساختار کلاس:

این کلاس شامل چندین استک برای مدیریت فرآیند تولید کد است: operand\_stack برای ذخیره‌سازی عملگرها، code\_stack برای ذخیره کدهای تولید شده، shape\_stack برای ابعاد ورودی، و aux\_stack برای ذخیره مقادیر اضافی (مثل پارامترهای تابع فعال‌سازی و غیره).

متغیر non\_operands لیستی از مقادیر غیر عملگری است که در کد به صورت خاص پردازش می‌شوند، مانند layer, activation, input\_shape, و غیره.

متدهای کلیدی:

is\_operand: این متد بررسی می‌کند که آیا یک آیتم عملگر است یا نه. اگر آیتم در لیست non\_operands نباشد، به عنوان عملگر در نظر گرفته می‌شود.

generate\_initial: این متد برای تولید کد اولیه برای وارد کردن کتابخانه‌های مورد نیاز و تعریف کلاس شبکه استفاده می‌شود.

generate\_instance: این متد برای تولید بخش‌هایی از کد که شامل راه‌اندازی و استفاده از نمونه‌ها می‌باشد، نوشته می‌شود. در صورت نیاز، این متد داده‌های آموزشی را تولید می‌کند و مدل را آموزش و ارزیابی می‌کند.

generate\_code: این متد کد اصلی شبکه را تولید کرده و آن را به فایل ذخیره می‌کند. این متد به‌طور اتوماتیک بر اساس دستورات ورودی، کدهای مربوط به شبکه عصبی را تولید می‌کند.

عملیات‌ها و کدهای مربوط به هر قاعده گرامری:

generate\_layer: این متد لایه‌های شبکه را با استفاده از نوع لایه و تعداد واحدها تولید می‌کند.

generate\_activation: این متد تابع فعال‌سازی لایه‌ها را مشخص می‌کند.

generate\_training: این متد کدهای مربوط به آموزش مدل، از جمله تنظیمات بهینه‌ساز، تابع خطا، و تعداد دوره‌ها را تولید می‌کند.

generate\_metric\_choice: این متد انتخاب معیارهای ارزیابی مدل را انجام می‌دهد.

generate\_evaluate: این متد کدهای مربوط به ارزیابی مدل را تولید می‌کند.

generate\_visualize: این متد برای تجسم شبکه و عملکرد آن بر روی یک گرید استفاده می‌شود.

generate\_dataset: این متد برای بارگذاری و پردازش داده‌ها (از جمله مجموعه داده‌های MNIST یا داده‌های سفارشی) استفاده می‌شود.

generate\_begin\_scope\_operator و generate\_end\_scope\_operator: این متدها برای مدیریت آغاز و پایان محدوده‌ها در گرامر استفاده می‌شوند.

مدیریت قوانین تولید کد:

generated\_rules: این متغیر برای ردیابی قوانینی است که کد برای آنها تولید شده است. به عنوان مثال، اگر کد مربوط به آموزش مدل یا ارزیابی تولید شده باشد، این قوانین به لیست generated\_rules اضافه می‌شوند.کد بر اساس دستوراتی که در آرایه وارد می‌شوند به صورت خودکار تولید می‌شود. این دستورها ممکن است شامل اطلاعات مربوط به لایه‌ها، تابع فعال‌سازی، اندازه داده‌ها و سایر تنظیمات شبکه عصبی باشند. تا در instance ساخته شده فقط توابعی که جنریت شدند صدا زده شوند.

# File Structure

## src:

## Input

نمونه فایل های ورودی شامل یک فایل با پسوند .deep هستند.

## Output

فایل به عنوان خروجی یک فایل png از ast و یک فایل پایتون مربوط به کد جنریت شده تولید میکند

## Code

فایل دارای یک فولدر Code شامل deepDSLCodeGenerator و deepDSLCustomLinstener است.

## gen

دارای فایل های configure و generate شده ANTLR است.

## Repository

حاوی فایل های تولید و نمایش AST است.

## Grammar

گرامر پروژه در این فولدر است.

همچنین فایل main.py برای ران کردن پروژه نهایی و requirements.txt برای نمایش لیست پکیج‌هایی که در پروژه خود از آنها استفاده میکنیم است.